Brevemente, ¿qué es y para qué sirve la característica de JAVA llamada Introspección? ¿y la característica de JAVA llamada Reflexión? ¿Qué paquete java ofrece esas características?

Introspección (Introspection): Es la capacidad de un programa para examinar la estructura de sus propios objetos en tiempo de ejecución (conocer sus métodos, campos y propiedades). Sirve principalmente para consultar las capacidades de una clase sin modificarla.

Reflexión (Reflection): Va un paso más allá. Es la capacidad de un programa para examinar y modificar el comportamiento en tiempo de ejecución. Permite instanciar clases dinámicamente, invocar métodos, modificar valores de variables (incluso privadas) y alterar la lógica del programa al vuelo.

Paquetes de Java: Las herramientas para reflexión residen en el paquete java.lang.reflect, mientras que la introspección específica para componentes (JavaBeans) se apoya también en el paquete java.beans.

¿Cuáles son los principales métodos que se usan en Introspección?

• Introspector.getBeanInfo(Class<?> beanClass): Obtiene toda la información de un componente.
• BeanInfo.getPropertyDescriptors(): Devuelve las propiedades (atributos con getters/setters).
• PropertyDescriptor.getReadMethod(): Obtiene el método getter.
• PropertyDescriptor.getWriteMethod(): Obtiene el método setter.

¿Cuáles son los principales métodos que se usan en Reflexión?

• Class.forName(String className): Carga una clase dinámicamente por su nombre.
• Class.getDeclaredFields() / getDeclaredMethods(): Obtienen todos los atributos o métodos de la clase (incluyendo privados).
• Method.invoke(Object obj, Object... args): Ejecuta un método dinámicamente.
• Field.set(Object obj, Object value): Asigna un valor a un atributo.
• Constructor.newInstance(Object... initargs): Crea una nueva instancia de la clase.

Busca un ejemplo sencillo de uso.

```
import java.lang.reflect.Field;
import java.lang.reflect.Method;

public class EjemploReflexion {
  public static void main(String[] args) {
    try {

            Class<?> miClase = Class.forName("Persona");

            Object personaInstance =
miClase.getDeclaredConstructor().newInstance();

            Field nombreField =
miClase.getDeclaredField("nombre");
            nombreField.setAccessible(true);
            nombreField.set(personaInstance, "Carlos");

            Method saludarMethod =
miClase.getDeclaredMethod("saludar");
            saludarMethod.invoke(personaInstance);

        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}

class Persona {
    private String nombre;

    public void saludar() {
        System.out.println("Hola, mi nombre es " + nombre);
    }
}
```

¿Qué harías si tuvieras el nombre de una clase como “literal” (con su paquete incluido) y necesitas crear un objeto de esa clase en tiempo de ejecución?

1. Cargar la clase usando Class<?> clf = Class.forName("com.empresa.modelo.Usuario");
2. Obtener el constructor deseado (por ejemplo, el vacío): Constructor<?> cons = clf.getDeclaredConstructor();
3. Crear el objeto: Object miObjeto = cons.newInstance();

¿Puedes invocar un método de un objeto sin conocerlo en tiempo de compilación? ¿Cómo?

Sí obteniendo la referencia al método a través del objeto Class y llamando a su función de ejecución dinámica.

```
// 1. Obtienes el método por su nombre string y los tipos de sus parámetros
Method metodo = objeto.getClass().getDeclaredMethod("nombreDelMetodo", String.class);

// 2. Lo invocas pasando la instancia del objeto y los argumentos requeridos
metodo.invoke(objeto, "Parámetro enviado");
```

¿Qué riesgos o desventajas crees que tiene usar reflexión en exceso?

• Pérdida de rendimiento (Performance Overhead): La resolución dinámica obliga a la JVM a realizar comprobaciones de seguridad y búsquedas en la estructura de clases que no se pueden optimizar en la compilación, haciéndolo mucho más lento.
• Vulnerabilidad de seguridad: Permite romper la encapsulación de datos (acceder a elementos privados con setAccessible(true)), lo que puede comprometer la integridad y el diseño del software.
• Errores en Tiempo de Ejecución (Runtime Errors): Al saltarse la verificación estática del compilador, cualquier error tipográfico en el nombre de una clase o método provocará excepciones destructivas en producción (ClassNotFoundException, NoSuchMethodException) en lugar de fallar al compilar.
• Dificultad de mantenimiento: El código se vuelve abstracto, difícil de leer y rompe las herramientas automatizadas de refactorización de los IDEs.

Dentro de acceso a datos (2º DAM), indícame 3 ejemplos de proyectos populares en los que se esté utilizando la introspección y la reflexión.

Hibernate ORM (y Spring Data JPA), Jackson / Gson (Mapeadores JSON/XML), Spring Framework (Inyección de Dependencias)